BRANCH COVERAGE TEST CASE GENERATION USING GENETIC ALGORITHM AND HARMONY SEARCH

HOSEIN ABEDINPOURSHOTORBAN

UNIVERSITI TEKNOLOGI MALAYSIA

BRANCH COVERAGE TEST CASE GENERATION USING GENETIC ALGORITHM AND HARMONY SEARCH

HOSEIN ABEDINPOURSHOTORBAN

A dissertation submitted in partial fulfilment of the
requirements for the award of the degree of
Master of Science (Computer Science)

Faculty of Computing
Universiti Teknologi Malaysia

JANUARY 2015

I strongly dedicated this dissertation to my beloved parents for their supports, encouragement and love.

ACKNOWLEDGEMENT

I would like to thank my supervisor **Assoc. Prof. Dr. Dayang Norhayati Abang Jawawi**, who has been an invaluable friend and mentor. Her gift for conceptualization, her enduring encouragement, and her practical advice have been an inestimable source of support for me during this research.

Besides, my endless gratitude goes to my parents for their strong moral and emotional support. I would also like to thank my dear friend Nur Fatimah As'Sahra who shared this journey with me, and always supported me.

ABSTRACT

Due to the vital role of software in the modern world, there is a great demand for reliability, and it can be achieved through the process of testing. White-Box testing is one of the testing methods that aims to uncover errors of coding by investigating the internal structure of the software. Moreover, generation of test cases for White-Box testing of software can be done manually or automatically. However, due to possible mistakes and expenses of manual test case generation, trend is toward making this activity automatic. So far, proposed techniques for automatic test case generation are mostly based on Genetic Algorithm (GA). However, existing GA techniques are quite slow and unable to achieve full coverage when it comes to test case generation for complex software with a wide range of inputs. Thus, in this research an improved fitness function is proposed based on Control Dependence Graph (CDG) and branch distance that can improve the speed and coverage of test cases generation by the means of evolutionary algorithms like GA. Also, a GA-based branch coverage test case generation technique is proposed in this research that takes advantage of our proposed fitness function, and comparison results based on two benchmark case studies show that our proposed technique outperforms the original CDG technique in speed and coverage of test case generation. In addition, we evaluated our proposed fitness function with harmony search algorithm (HS), which is a more recent optimization algorithm compared to GA, and find out that HS outperforms GA in speed of test case generation for branch coverage of software code.

ABSTRAK

Oleh kerana perisian memainkan peranan yang penting dalam dunia moden, terdapat permintaan yang besar terhadap kemampuannya, dan ia boleh dicapai melalui proses ujian. Ujian Kotak Putih merupakan salah satu kaedah ujian yang bertujuan untuk mendedahkan kesilapan pengekodan dengan menyiasat struktur dalaman perisian. Selain itu, generasi kes-kes ujian untuk ujian Kotak Putih terhadap perisian boleh dilakukan secara manual ataupun secara automatik. Namun, disebabkan oleh kesilapan dan penggunaan ujian manual generasi kes, haluan ke arah menjadikan aktiviti ini automatik. Setakat ini, teknik yang dicadangkan untuk kes ujian generasi automatik kebanyakannya berdasarkan Algoritma Genetik (GA). Namun, teknik GA sedia ada agak perlahan dan tidak dapat mencapai liputan sepenuhnya apabila ia digunakan untuk menguji generasi kes untuk perisian kompleks dengan pelbagai input. Oleh itu, dalam kajian ini fungsi kecergasan yang lebih baik adalah dicadangkan berdasarkan Kawalan Kebergantungan Graf (CDG) dan jarak cawangan yang boleh meningkatkan kelajuan dan liputan generasi kes-kes ujian dengan cara-cara evolusi algoritma seperti GA. Tambahan lagi, satu cabang ujian liputan teknik penjanaan kes berdasarkan GA dicadangkan dalam kajian ini yang mengambil kesempatan daripada fungsi kecergasan yang kami dicadangkan, dan keputusan perbandingan berdasarkan dua kajian kes penanda aras menunjukkan bahawa teknik yang kami cadangkan melebihi prestasi teknik CDG asal dalam kelajuan dan liputan ujian generasi kes. Selain itu, kami menilai fungsi kecergasan kami dengan algoritma pencarian harmoni (HS), yang merupakan algoritma pengoptimuman yang lebih terkini berbanding GA, dan kami mengetahui bahawa HS melebihi prestasi GA dalam kelajuan ujian generasi kes untuk liputan cawangan kod perisian.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACO	-	Ant Colony Optimization
AI	-	Artificial Intelligence
CDG	-	Control Dependence Graph
CFG	-	Control Flow Graph
DE	-	Deferential Evolution
GA	-	Genetic Algorithm
HMCR	-	Harmony Memory Consideration Rate
HMS	-	Harmony Memory Size
HS	-	Harmony Search
PAR	-	Pitch Adjustment Rate
SRS	-	Software Requirements Specifications
SUT	-	Software Under Testing
V&V	-	Verification and Validation

CHAPTER 1

INTRODUCTION

1.1 Overview

Nowadays, software has various applications from computers and mobile phones to the airbag control systems and military (Qian and Zheng, 2009). Moreover, a considerable amount of software is used by corporations, which have major effects on their business (Sharma and Kumar, 2012). Also, prediction for the next decade is an exponential increase in software usage (Liggesmeyer and Trapp, 2009). Therefore, software has an enormous influence on our lives and play crucial part in it (Lyu, 2007).

Caused by reasons mentioned above, the goal of the software industry is delivery of good quality software to the user. To achieve this goal, software testing is vital. Testing ensures meeting of user requirements and specifications. However, in software testing there are a lot of underlying issues that need to be considered. Tackling of these issues demand time, effort and cost (Sharma *et al.*, 2013). In software development, more than 50% of cost belongs to testing (Samuel *et al.*, 2007).

As a result of dramatic rise in the size and complexity of software, testing is an indispensable activity of software development (Humphrey, 2001). In other words, testing evaluate the software system execution to confirm whether it acts as intended. Testing has

industrial usage for quality assurance by inspecting the execution of the software and providing proper feedback on software behavior (Bertolino, 2007).

Testing is defined as the system execution toward the purpose of finding errors in the system. It contains a broad range of various approaches with different motivations and purpose. The test quality is measured by its ability to find errors. Therefore, tests must be based on the requirements that domain experts defined for a system (Stahl and Voelter, 2006).

Software testing is a widely used term for a wide spectrum of different activities, from the unit testing of code by the programmers, to the validation of a large system by customer (acceptance testing), to the run-time monitoring of a service-oriented application. Test cases can be generated for different objectives, such as measuring possession of user requirements, or measuring robustness to heavy load situations or to invalid inputs, or evaluating given attributes, such as usability or performance, or estimating the trustworthiness of operations, etc. Besides, the testing can be carried based on a controlled formal process, requiring exact documentation and planning, or rather informally (exploratory testing) (Bertolino, 2007).

During software development, testing is one of the crucial activities and need to be performed precisely. According to a study conducted by the National Institute of Standard & Technology, software bugs cost the United States economy around $59.5 billion a year, with one-third of this value being attributed to the poor software testing (Silva and van Someren, 2010). Therefore, creation of a relevant subset of test cases is of great importance. Faults should be exposed by the test cases which are used to examine the software under testing (SUT) and the test cases should be based on possible inputs (Gupta and Rohil, 2013). Quality of the testing is directly related and affected by the set of test cases that are generated to perform testing (Gupta and Rohil, 2008).

In the process of software development, generation of test cases is mostly a manual activity and the testers are responsible for doing it. Consequently, this part of software

development is extremely difficult, laborious, and expensive (McMinn, 2004). Automation of test case generation can improve the efficiency of software testing and certainly can reduce the designing expense of software, reduction of the time needed for development of software, and significantly improve the quality of software (Khamis *et al.*, 2012). Mainly, automatic test case generation is done by test data generation methods that take advantage of soft computing algorithms like Genetic Algorithm (GA) (Sthamer *et al.*, 2002).

Software testing is based on three strategies: white-box testing, black-box testing, and gray-box testing. White-box testing is also known as structural testing that tests the SUT to gain as much coverage of the code as possible (Panchapakesan *et al.*, 2013). Black box testing is known as functional testing, which tests the SUT to make sure that the software is loyal to the specifications (Panchapakesan *et al.*, 2013). Grey-box testing is also known as model-based testing, which tests the SUT using generated test cases from design models.

Among software testing techniques, White-Box testing of software unit is used to test components of software system and examine the internal structure and coding of the program. The goal is to execute every instruction in the code at least once (Myers et al., 2011). In principle, unit testing is an important activity during testing and has a crucial role in finding bugs. In practice, unit testing is so costly and difficult and rarely done properly. As a consequence, lots of software bugs remain uncovered (Godefroid *et al.*, 2005). In unit testing, test cases generally are generated based on some predefined testing criteria, for instance, path coverage or branch coverage (Baresel *et al.*, 2002).

As we discussed importance of White-Box testing, the focus of this research is on the test case generation for White-Box testing of software. But, complexity of finding test cases to satisfy testing criteria from wide range of software input domain causes most of the proposed methods in this area take advantage of soft computing algorithms which are designed for solving nondeterministic hard problems. Nowadays, many techniques have been suggested to address this issue based on GA. The reason for choosing GA among a broad range of soft computing algorithm is the ability of GA to avoid local minima and

finding good solution for hard problems. Therefore, the primary focus of this research is on test case generation for with box testing of software by the means of GA.

1.2 Problem Background

In the past, automatic test data generation methods have been used for the simple programs using simple test criteria. Therefore, random test generation was sufficient for these problems. Nevertheless, it seems impossible that random techniques would be able to perform well on realistic and complicated test-generation problems, which usually needs an intensive manual effort (Michael *et al.*, 2001)

In recent years, usage of metaheuristic techniques for the automatic generation of test data has been interesting for researchers. Existing random-based methods for automation of the test case generation have limitations on the complexity and the size of software. Therefore, metaheuristic search techniques are introduced to software testing to solve these problems. Metaheuristic search techniques are high-level frameworks, which use heuristics to find solutions for complex problems at a reasonable computational cost. To date, metaheuristic search techniques have been used for automating test data generation for structural and functional testing (McMinn, 2004).

Three metaheuristic algorithms have been used for software testing. First one is “Hill Climbing" that is a well-known local search algorithm. Hill Climbing enhances a randomly selected solution by investigating the neighborhood of the solution, if the algorithm discovers a better solution, then better one replaces the existing solution. Second one is “simulated annealing”, which in principle is similar to Hill Climbing. But, by the chance of accepting poorer solutions, Simulated Annealing is less restricted compared to Hill Climbing in movement around the search space (McMinn, 2004). However, the mentioned algorithms are only useful in local searching and demonstrate poor performance for global searching and cannot find test data for sophisticated application. Therefore,

researchers emigrate to GA, which is a well-known algorithm for global searching and will be discussed in detail in following.

Nowadays those techniques using metaheuristic methods are more advanced. GA is the base of the majority of them, and it is proved that GA can at least perform like random algorithms, yet it shows better performance in most cases (Briand *et al.*, 2002).

The first group of methods used conventional GA like, Pargas *et al.* (1999) which presents a goal-oriented technique for automatic test-data generation that uses a GA and Control Dependence Graph (CDG) of software, and this algorithm is capable to be executed in parallel on multiple processors to reduce the execution time. In another approach an automatic test case generation method for structural testing of software is proposed by Girigis (2005) that takes advantage of using GA and data flow dependencies of the program using this algorithm they improve the effectiveness of test cases. Similarly, another technique for structural testing of software by the means of GA is proposed by Alzabidi *et al.* (2009). A path coverage criterion is used in this technique for testing structure of software and fitness function of GA is improved in this technique. In another work, Srivastava and Kim (2009) proposed a method that by recognizing most critical path in software code can improve efficiency of software testing.

Other techniques are combined GA with other search-based algorithms. Srivastava *et al.* (2008) proposed a hybrid method for test case generation based on path coverage criteria combined with Ant Colony Optimization algorithm and this algorithm prevent trapping in infinite loop while generating test cases. Another hybrid technique proposed by Zhang and Wang (2011) which takes advantage of "simulated anneal algorithm" combined by GA for testing path in the program, and this algorithm has better speed in covering objective path.

As we discussed earlier, there are two types of GA methods. The first group is using conventional GA, and the second group is using hybrid GA. However, there are some issues in each group, and we are going to mention them overly. First group suffer

from slow convergence and is not able to fully cover a big application in finite time. Moreover, the second group suffers from immature convergence that leads to the generation of less effective test cases.

According to Pachauri and Srivastava (2013) there are three techniques using GA for branch coverage of software code and the proposed a technique by Pargas *et al.* (1999) is the latest one in this area. Although, research has been continuing, but most of the researchers are focusing on improvement of proposed technique by Pargas *et al.* (1999). For instance, Miller *et al*. (2006) proposed a method to deal with Boolean and enumerated types. Furthermore, Arcuri (2010) focused on the effects of branch distance normalization on fitness evaluation. In addition, Pachauri and Srivastava (2013) evaluated the impact of branch selection on the speed of coverage. Although, the fitness function has a significant effect on speed and coverage, there is no research on enhancing the proposed fitness function by Pargas *et al*. (1999).

Majority of methods for test case generation are manually, and only a small number of techniques are proposed for automated test generation. These automated techniques are based on random, structural or path-oriented, analysis-oriented, and goal or branch-oriented test-data generation. However, there are some limitations in these methods. For instance, in random technique, lack of information about the objective of testing causes generation of a big number of test cases and usually cannot satisfy the objective of testing. Moreover, generation of path in a structural or path-oriented method sometimes is impossible because of the inability of the generator to find an input to traverse the path. Besides, analysis-oriented generators are highly relied upon the accuracy of design and need lots of modeling using tools, which is impossible for some software systems. Hence, the best method is branch-oriented technique due to the lower number of generated test cases compared to other methods (Miller *et al.*, 2006).

Pargas *et al*. (1999) achieved limited success in the generation of test cases for branch coverage of small programs. However, there are still challenges in terms of improving speed and coverage of this method, and there is need for researches to be conducted in this area. Also, there are many optimization algorithms that have not been

used for White-Box test case generation. Harmony Search (HS) is one of the recent optimization algorithms that share many characteristic with GA and does not suffer from slow convergence and recently have been used successfully in other areas of software testing like interaction test data generation (Alsewari and Zamli, 2011).

1.3 Problem Statement

In the effort to improve testing, a number of methods have been proposed to automate the test case generation. However, the automation of test data generation is still a topic under research. Recently GA have been used to automate the testing process (Silva and van Someren, 2010). However, generating test cases to cover the code of big software using current GA techniques is time-consuming, and for some cases even impossible due to the limitation of testing time and speed of contemporary computers.

Also, there are many optimization algorithms that have not been exploited in the area of White-Box testing, and most of the researchers are focused on GA. Therefore, there is need for researches to be conducted on evaluation of the performance of other optimization algorithms in this area. For instance, HS has been more successful than GA in other areas of software testing like interaction test data generation but has not been used for White-Box testing.

Therefore, the research question posed is “how can we improve the speed and the coverage of test case generation for branch coverage of software code by enhancing an existing technique?”

1.4 Research Aim

The primary aim of this research is to identify limitations of GA-Based software testing techniques. Also, to propose an enhanced method that would improve the performance (speed and coverage) of GA for branch coverage test case generation, by improving Pargas *et al*. (1999) technique. In addition, comparing the performance of GA with another more recent optimization technique called HS for branch coverage test case generation.

1.5 Research Objectives

In order to achieve the above-mentioned aim, the objectives of this research are as follows:

- To identify contemporary limitations of existing GA-Based software testing techniques.
- To improve performance of an existing test case generation technique based on GA in terms of speed and coverage of test case generation.
- To compare the proposed technique with the original one (an existing technique based on GA) based on the speed and the coverage of test case generation..
- To compare the performance of GA with HS for branch coverage test case generation.

1.6 Scope of the Study

This study is intended for several small to big-sized software applications that require accuracy and consistency in their system functionality before they are released to

the market, i.e. those where testing the accuracy for quality is vital. The following are significant:

- This study only focuses on using of GA and HS for automatic test case generation.
- This study only focuses on the system unit level (White-Box) test case generation, leaving out the rest areas.
- In this study we chose an existing technique using GA and try to enhance it in terms of speed of test case generation.
- The enhanced test case generation technique is compared with the original one, an already existing technique using GA.
- The performance of HS is investigated in the area of software testing.

1.7 Significance of the Study

Software testing is very expensive and laborious process, we intend to help the software industry to reduce expenses and effort that is needed for software testing. In other hand, because testing of the system based on human testing is error prone area we want to reduce the possible fault in software testing with proposing automated approach. Also, it will be beneficial to the researchers those who are interested in carrying out their research in the area of software testing.

1.8 Dissertation Organization

This research is made up of six chapters. In Chapter 1, we discussed the research introduction, problem background, problem statement and objectives of the study. Similarly, Chapter 2 presents an overview of software testing with the focus on White-Box testing, GA-based test case generation as well as the literature review of the study. In Chapter 3, we explained the research methodology in sequence of phases. Moreover,

Chapter 4 presents the proposed technique to improve the performance of test case generation for branch coverage of software code.

Furthermore, in Chapter 5, we compare the proposed technique and original technique based on speed and coverage. Also, we evaluate the performance of (HS + our proposed fitness function) against the proposed GA for branch coverage test case generation. In addition, Chapter 6 presents the study summary, contributions, and future works.

REFERENCES

Afzal, W. 2007. Metrics in software test planning and test design processes. *MSE-2007, 2*, pp 111.

Aljahdali, S. H., Ghiduk, A. S., and El-Telbany, M. 2010. *The limitations of genetic algorithms in software testing.* Computer Systems and Applications (AICCSA), 2010 IEEE/ACS International Conference on, pp 1-7.

Alsewari, A., and Zamli, K. Z. 2011. *Interaction test data generation using harmony search algorithm.* Industrial Electronics and Applications (ISIEA), 2011 IEEE Symposium on, pp 559-564.

Alzabidi, M., Kumar, A., and Shaligram, A. 2009. Automatic Software structural testing by using Evolutionary Algorithms for test data generations. *IJCNS International Journal of Computer Science and Network Security, 9*(4), pp 390-395.

Arcuri, A. 2010. *It does matter how you normalise the branch distance in search based software testing.* Software Testing, Verification and Validation (ICST), 2010 Third International Conference on, 205-214.

Ball, T. 1999. *The concept of dynamic analysis.* Software Engineering—ESEC/FSE'99, pp 216-234.

Baresel, A., Sthamer, H., and Schmidt, M. 2002. *Fitness Function Design To Improve Evolutionary Structural Testing.* GECCO, pp 1329-1336.

Bertolino, A. 2007. *Software testing research: Achievements, challenges, dreams.* Future of Software Engineering, 2007. FOSE'07, pp 85-103.

Bertolino, A., and Marchetti, E. 2005. A brief essay on software testing. *Software Engineering, 3rd edn. Development process, 1*, pp 393-411.

Blackwell, B. M., Collen, J. P., Guzman Jr, L. R., Irwin, A. G., Kokke, B. M., and Lindsay, J. D. 2009. Testing tool comprising an automated multidimensional traceability matrix for implementing and validating complex software systems: Google Patents.

Boehm, B., and Basili, V. R. 2007. Software defect reduction top 10 list. *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research, 34*(1), pp 75.

Briand, L. C., Feng, J., and Labiche, Y. 2002. *Using genetic algorithms and coupling measures to devise optimal integration test orders.* Proceedings of the 14th international conference on Software engineering and knowledge engineering, pp 43-50.

Buxton, J. N., and Randell, B. 1970. *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee*: NATO Science Committee; available from Scientific Affairs Division, NATO.

Cai, X., and Lyu, M. R. 2005. *The effect of code coverage on fault detection under different testing profiles.* ACM SIGSOFT Software Engineering Notes, pp 1-7.

Craig, R. D., and Jaskiel, S. P. 2002. *Systematic software testing*: Artech House.

Geem, Z. W., Kim, J. H., and Loganathan, G. 2001. A new heuristic optimization algorithm: harmony search. *Simulation, 76*(2), pp 60-68.

Ghiduk, A. S. 2014. Automatic generation of basis test paths using variable length genetic algorithm. *Information Processing Letters*.

Girgis, M. R. 2005. Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm. *J. UCS, 11*(6), pp 898-915.

Glinz, M. 2011. A glossary of requirements engineering terminology. *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version, pp 1*.

Godefroid, P., Klarlund, N., and Sen, K. 2005. *DART: directed automated random testing.* ACM Sigplan Notices, pp 213-223.

Gupta, N. K., and Rohil, M. K. 2008. *Using genetic algorithm for unit testing of object oriented software.* Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on, pp 308-313.

Gupta, N. K., and Rohil, M. K. 2013. *Improving GA based automated test data generation technique for object oriented software.* Advance Computing Conference (IACC), 2013 IEEE 3rd International, pp 249-253.

Harman, M., and McMinn, P. 2010. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *Software Engineering, IEEE Transactions on, 36*(2), pp 226-247.

Humphrey, W. S. 2001. The future of software engineering: I. *Watts New Column, News at SEI, 4*(1).

Jin, R., Jiang, S., and Zhang, H. 2011. *Generation of test data based on genetic algorithms and program dependence analysis.* Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2011 IEEE International Conference on, pp 116-121.

Khamis, A. M., Girgis, M. R., and Ghiduk, A. S. 2012. Automatic software test data generation for spanning sets coverage using genetic algorithms. *Computing and Informatics, 26*(4),pp 383-401.

Khan, M. 2011. Different Approaches to White Box Testing Technique for Finding Errors. *International Journal of Software Engineering & Its Applications, 5*(3).

Khan, M. E., and Khan, F. 2012. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. *International Journal of Advanced Computer Sciences and Applications, 3*(6), pp 12-15.

Liggesmeyer, P., and Trapp, M. 2009. Trends in embedded software engineering. *Software, IEEE, 26*(3), pp 19-25.

Lin, J.-C., and Yeh, P.-L. 2001. Automatic test data generation for path testing using GAs. *Information Sciences, 131*(1), pp 47-64.

Lyu, M. R. 2007. *Software reliability engineering: A roadmap.* 2007 Future of Software Engineering, pp 153-170.

Machado, P., Vincenzi, A., and Maldonado, J. C. 2010. Software testing: an overview. In *Testing Techniques in Software Engineering* (pp. 1-17): Springer.

Marick, B. 1999. New models for test development. *Testing Foundations*.

McMinn, P. 2004. Search-based software test data generation: a survey. *Software Testing, Verification and Reliability, 14*(2), pp 105-156.

Michael, C. C., McGraw, G., and Schatz, M. A. 2001. Generating software test data by evolution. *Software Engineering, IEEE Transactions on, 27*(12), pp 1085-1110.

Miller, J., Reformat, M., and Zhang, H. 2006. Automatic test data generation using genetic algorithm and program dependence graphs. *Information and Software Technology, 48*(7), pp 586-605.

Misurda, J. 2011. *Efficient branch and node testing.* University of Pittsburgh.

Myers, G. J., Sandler, C., and Badgett, T. 2011. *The art of software testing*: John Wiley & Sons.

Pachauri, A., and Srivastava, G. 2013. Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism. *Journal of Systems and Software, 86*(5), pp 1191-1208.

Panchapakesan, A., Abielmona, R., and Petriu, E. 2013. *Dynamic white-box software testing using a recursive hybrid evolutionary strategy/genetic algorithm.* Evolutionary Computation (CEC), 2013 IEEE Congress on, pp 2525-2532.

Pargas, R. P., Harrold, M. J., and Peck, R. R. 1999. Test-data generation using genetic algorithms. *Software testing verification and reliability, 9*(4), pp 263-282.

Qian, H.-m., and Zheng, C. 2009. *A Embedded Software Testing Process Model.* Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on, pp 1-5.

Rajappa, V., Biradar, A., and Panda, S. 2008. *Efficient software test case generation using genetic algorithm based graph theory.* Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on, pp 298-303.

Rakitin, S. R. 2001. *Software verification and validation for practitioners and managers*: Artech House, Inc.

Rathore, A., Bohara, A., Prashil, R. G., Prashanth, T., and Srivastava, P. R. 2011. *Application of genetic algorithm and tabu search in software testing.* Proceedings of the Fourth Annual ACM Bangalore Conference, 23.

Ribeiro, J. C. B., Rela, M. Z., and de Vega, F. F. 2008. *A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object-oriented software.* Proceedings of the 3rd international workshop on Automation of software test, pp 85-92.

Roger, S. 2005. Software Engineering a Practitioner's Approach. *McGrow-Hill International Edition.*

Samuel, P., Mall, R., and Kanth, P. 2007. Automatic test case generation from UML communication diagrams. *Information and software technology, 49*(2), pp 158-171.

Sharma, C., Sabharwal, S., and Sibal, R. 2013. A Survey on Software Testing Techniques using Genetic Algorithm.

Sharma, M. A. K., and Kumar, D. 2012. User Acceptance of Desktop Based Computer Software Using UTAUT Model and addition of New Moderators.

Shull, F., Basili, V., Boehm, B., Brown, A. W., Costa, P., Lindvall, M., et al. 2002. *What we have learned about fighting defects.* Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on, pp 249-258.

Silva, L. S., and van Someren, M. 2010. *Evolutionary testing of object-oriented software.* Proceedings of the 2010 ACM Symposium on Applied Computing, pp 1126-1130.

Sofokleous, A. A., and Andreou, A. S. 2008. Automatic, evolutionary test data generation for dynamic software testing. *Journal of Systems and Software, 81*(11), 1883-1898.

Srivastava, P. R., and Kim, T.-h. 2009. Application of genetic algorithm in software testing. *International Journal of software Engineering and its Applications, 3*(4), pp 87-96.

Srivastava, P. R., Ramachandran, V., Kumar, M., Talukder, G., Tiwari, V., and Sharma, P. 2008. *Generation of test data using meta heuristic approach.* TENCON 2008-2008 IEEE Region 10 Conference, pp 1-6.

Stahl, T., and Voelter, M. 2006. *Model-driven software development*: John Wiley & Sons Chichester.

Sthamer, H., Wegener, J., and Baresel, A. 2002. *Using evolutionary testing to improve efficiency and quality in software testing.* Proceedings of the 2nd Asia-Pacific Conference on Software Testing Analysis and Review (AsiaSTAR).

Tracey, N., Clark, J., Mander, K., and McDermid, J. 1998. *An automated framework for structural test-data generation.* Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on, pp 285-288.

Wang, X., and Yan, X. 2013. Global best harmony search algorithm with control parameters co-evolution based on PSO and its application to constrained optimal problems. *Applied Mathematics and Computation, 219*(19), pp10059-10072.

Xiang, W.-l., An, M.-q., Li, Y.-z., He, R.-c., and Zhang, J.-f. 2014. An improved global-best harmony search algorithm for faster optimization. *Expert Systems with Applications, 41*(13), pp5788-5803.

Yang, Q., Li, J. J., and Weiss, D. M. 2009. A survey of coverage-based testing tools. *The Computer Journal, 52*(5), pp 589-597.

Zhang, B., and Wang, C. 2011. *Automatic generation of test data for path testing by adaptive genetic simulated annealing algorithm.* Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on, pp 38-42.

Zhao, S.-Z., Suganthan, P. N., Pan, Q.-K., and Fatih Tasgetiren, M. 2011. Dynamic multi-swarm particle swarm optimizer with harmony search. *Expert Systems with Applications, 38*(4), pp 3735-3742.

Zhu, H., Hall, P. A., and May, J. H. 1997. Software unit test coverage and adequacy. *Acm computing surveys (csur), 29*(4), pp 366-427.